© 2015 МГУ/ВМК/СП

Лекция 0x11

4 апреля

© 2015 МГУ/ВМК/СП

Далее …

- Явные списки свободных блоков
- Раздельные списки свободных блоков
- Сборка мусора
- ***Типичные ошибки при работе с памятью***

© 2015 МГУ/ВМК/СП

Типичные ошибки при работе с памятью

- Разыменование дефектного указателя
- Чтение неинициализированной памяти
- Перезапись памяти
- Ссылки на не существующие переменные
- Многократное освобождение блоков памяти
- Ссылки на уже освобожденные блоки
- Ошибки, связанные с освобождением блоков

© 2015 МГУ/ВМК/СП

Операции в языке Си

Операторы	*Ассоциативность*
`() [] -> .`	слева направо
`! ~ ++ -- + - * & (type) sizeof`	справа налево
`* / %`	слева направо
`+ -`	слева направо
`<< >>`	слева направо
`< <= > >=`	слева направо
`== !=`	слева направо
`&`	слева направо
`^`	слева направо
`|`	слева направо
`&&`	слева направо
`||`	слева направо
`?:`	справа налево
`= += -= *= /= %= &= ^= != <<= >>=`	справа налево
`,`	слева направо

`*p->`

- `->`, `( )`, и `[ ]` имеют более высокий приоритет, чем `*` и `&`
- Унарные операции `+`, `-`, и `*` имеют больший приоритет, нежели их бинарные аналоги

Источник: «Керниган & Ричи» страница 55

© 2015 МГУ/ВМК/СП

Объявление Си указателей: самопроверка

`int *p`	`p` – указатель на `int`
`int *p[13]`	`p` – массив из `[13]` указателей на `int`
`int *(p[13])`	`p` – массив из `[13]` указателей на `int`
`int **p`	`p` – указатель на указатель на `int`
`int (*p)[13]`	`p` – указатель на массив из `[13]` элементов `int`
`int *f()`	`f` – функция, возвращающая указатель на `int`
`int (*f)()`	`f` – указатель на функцию, возвращающую `int`
`int (*(*f())[13])()`	`f` – функция возвращающая указатель на массив `[13]` указателей на функции, возвращающие `int`
`int (*(*x[3])())[5]`	`x` – массив `[3]` указателей на функции, возвращающие указатели на массив `[5]` элементов `int`

Источник: «Керниган & Ричи» раздел 5.12

© 2015 МГУ/ВМК/СП

Разыменование дефектного указателя

- Классическая ошибка при использовании функции `scanf`

```
int val;

...

scanf(“%d”, val);
```

© 2015 МГУ/ВМК/СП

Чтение неинициализированной памяти

- Ошибочное предположение, что память, полученная из кучи, предварительно была заполнена нулями

```
/* вычисляем y = Ax */
int *matvec(int **A, int *x) {
   int *y = malloc(N*sizeof(int));
   int i, j;

   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         y[i] += A[i][j]*x[j];
   return y;
}
```

© 2015 МГУ/ВМК/СП

Перезапись памяти

- Выделение памяти с неправильным определением размеров (в некоторых случаях)

```
int **p;

p = malloc(N*sizeof(int));

for (i=0; i<N; i++) {
   p[i] = malloc(M*sizeof(int));
}
```

Что будет, если такой код перенести с IA-32 на x86-64?

© 2015 МГУ/ВМК/СП

Перезапись памяти

- Ошибка диапазона

```
int **p;

p = malloc(N*sizeof(int *));

for (i=0; i<=N; i++) {
   p[i] = malloc(M*sizeof(int));
}
```

© 2015 МГУ/ВМК/СП

Перезапись памяти

- Не проверяется превышение максимального размера вводимой строки

```
char s[8];
int i;

gets(s);  /* читаем “123456789” со стандартного входа */
```

- Основа для реализации классической атаки «переполнение буфера»

© 2015 МГУ/ВМК/СП

Перезапись памяти

- Неправильное понимание адресной арифметики

```
int *search(int *p, int val) {

   while (*p && *p != val)
      p += sizeof(int);

   return p;
}
```

© 2015 МГУ/ВМК/СП

Перезапись памяти

- Использование указателя вместо объекта, на который он указывает

```
int *BinheapDelete(int **binheap, int *size) {
   int *packet;
   packet = binheap[0];
   binheap[0] = binheap[*size - 1];
   *size--;
   Heapify(binheap, *size, 0);
   return(packet);
}
```

© 2015 МГУ/ВМК/СП

Ссылки на не существующие переменные

- Автоматические локальные переменные перестают существовать после выхода из функции

```
int *foo () {
   int val;

   return &val;
}
```

© 2015 МГУ/ВМК/СП

Многократное освобождение блоков памяти

- Фу!

```
x = malloc(N*sizeof(int));
      // что-то делаем с x
free(x);

y = malloc(M*sizeof(int));
      // что-то делаем с y
free(x);
```

© 2015 МГУ/ВМК/СП

Ссылки на уже освобожденные блоки

- Фу-фу-фу!

```
x = malloc(N*sizeof(int));
   // что-то делаем с x
free(x);
   ...
y = malloc(M*sizeof(int));
for (i=0; i<M; i++)
   y[i] = x[i]++;
```

© 2015 МГУ/ВМК/СП

Ошибки, связанные с освобождением блоков (утечка памяти)

- Долгое, но верное «убийство» программы

```
foo() {
   int *x = malloc(N*sizeof(int));
   ...
   return;
}
```

© 2015 МГУ/ВМК/СП

Ошибки, связанные с освобождением блоков (утечка памяти)

- Освобождаем только часть сложно устроенной структуры данных

```
struct list {
   int val;
   struct list *next;
};

foo() {
   struct list *head = malloc(sizeof(struct list));
   head->val = 0;
   head->next = NULL;
   // создаем и что-то делаем с остальной частью списка
    ...
   free(head);
   return;
}
```

© 2015 МГУ/ВМК/СП

И что же со всем этим делать?

- Традиционный отладчик (gdb)
  - Удобно находить разыменование дефектных указателей
  - Совсем неудобно выявлять все остальные виды ошибок

При срабатывании ошибки крайне важна выдаваемая диагностика!

- Отладочная версия `malloc`
  - Обертка вокруг обычной функции `malloc`
  - Выявление ошибок в рамках функций `malloc` и `free`
    - Повреждение внутренних структур кучи в результате перезаписи
    - В некоторых случаях – многократное освобождение памяти
    - Утечки памяти
  - Не может выявлять все остальные ошибки
    - Перезаписи внутри выделенных блоков
    - Использование уже освобожденных блоков
    - …

© 2015 МГУ/ВМК/СП

И что же со всем этим делать?

- Некоторые реализации malloc уже содержат код дополнительных проверок
  - glibc malloc в ОС Linux: `setenv MALLOC_CHECK_ 2`
  - FreeBSD: `setenv MALLOC_OPTIONS AJR`
- Программные инструменты динамического анализа
  - valgrind (ОС Linux), Purify
    - Двоичная трансляция
  - Работают с исполняемы кодом программы
  - Может выявлять все ошибки, что и отладочный `malloc`
  - А также проверяет все обращения к памяти во время выполнения программы
    - Дефектные указатели
    - Перезапись памяти
    - Ссылка вне выделенных блоков памяти

- ***Дают детальную диагностику, если ошибка сработала***
- ***Нужны начальные данные, на которых ошибки будут проявляться***

© 2015 МГУ/ВМК/СП

И что же со всем этим делать?

- Автоматическая сборка мусора
  - Консервативный сборщик Boehm-GC для Си/Си++
- Статический анализ исходного кода
  - Легковесный анализ
    - Встраивается в среду разработки
  - Глубокий анализ: построение потоков управления, данных, межпроцедурный анализ всего кода программы, и т.д.
    - Встраивается в систему сборки

- ***Анализ всех возможных путей выполнения при произвольных входных данных***
- ***Для понимания диагностики требуется хорошая квалификация***

- На практике профессиональные разработчики программ комбинируют всё перечисленное
  - А также кое-что еще…

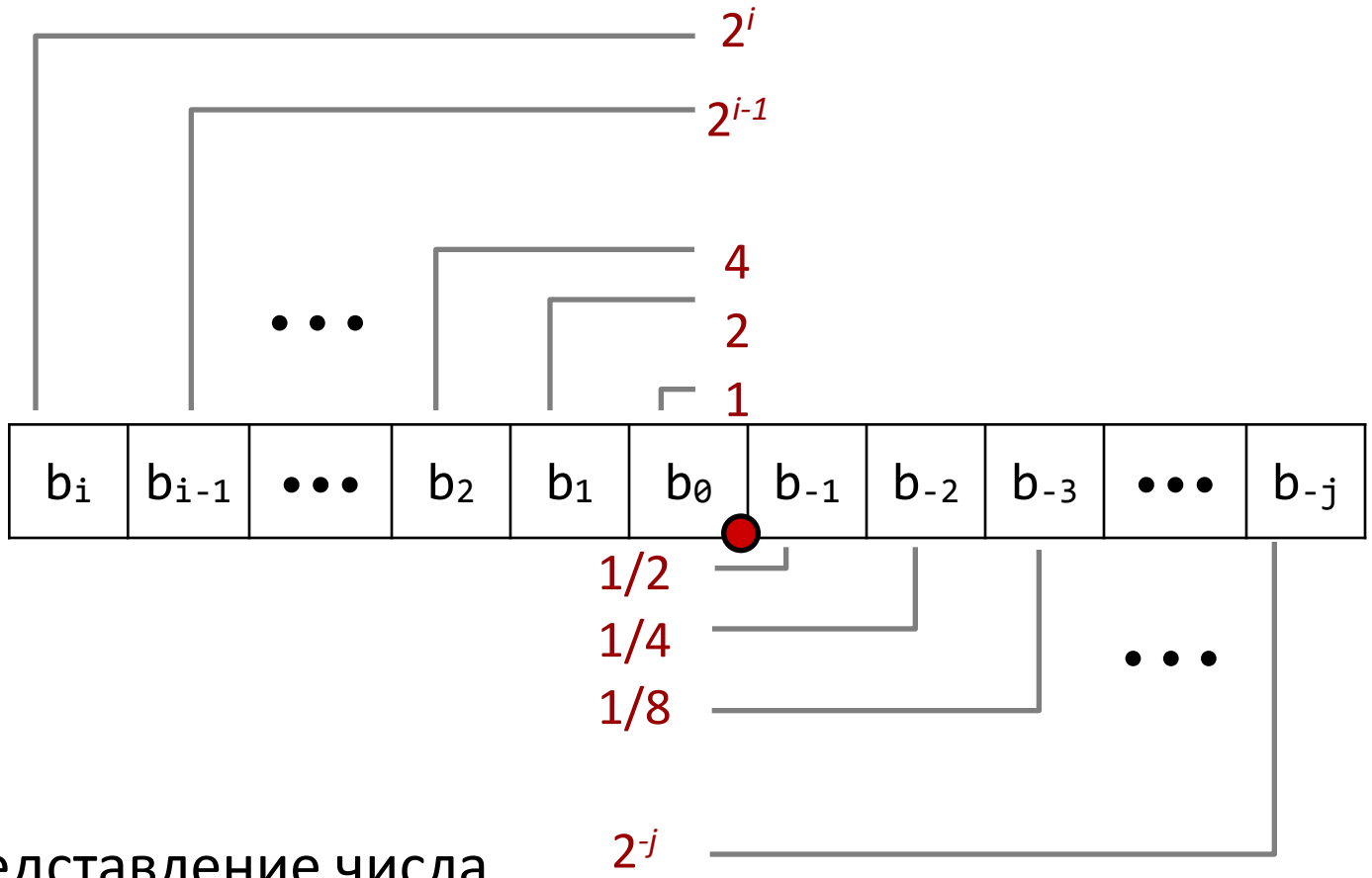
© 2015 МГУ/ВМК/СП

Далее ...

- ***Динамическая память***
  - ***Организация и управление***
  - ***Численные характеристики***
  - ***Управление свободными блоками***
  - ***Сборка мусора***
  - ***Типовые ошибки в Си программах при работе с памятью***
- ***Числа с плавающей точкой***
  - ***Представления для вещественных чисел***
    - ***Дробные двоичные числа***
    - ***Числа с плавающей точкой***
  - ***Сопроцессор x87***
    - ***Устройство***
    - ***Примеры программ***

© 2015 МГУ/ВМК/СП

Дробные двоичные числа



- Представление числа
  - Биты справа от “двоичной точки” представляют отрицательные степени 2
  - Точное представление для рациональных чисел вида : $\sum_{k=-j}^{i} b_k \times 2^k$

© 2015 МГУ/ВМК/СП

Примеры дробных двоичных чисел

Число	Представление
5 3/4	101.11_2
2 7/8	10.111_2
63/64	0.111111_2

- Деление на 2 может выполняться сдвигом вправо, …
- … а умножение на 2 – сдвигом влево
- Числа вида $0.11111\ldots_2$
  - На один «шаг» меньше чем `1.0`
  - Используется специальное обозначение `1.0 – ε`

© 2015 МГУ/ВМК/СП

Представимые рациональные числа

- Ограничение
  - Можно представить рациональные числа только вида $x/2^k$
  - Другие рациональные числа представляются повторяющимися группами бит

- Число Представление

1/3	$0.0101010101[01]\ldots_2$
1/5	$0.001100110011[0011]\ldots_2$
1/10	$0.0001100110011[0011]\ldots_2$

© 2015 МГУ/ВМК/СП

Представление чисел с плавающей точкой

- Численное представление

$$(-1)^s \times M \times 2^E$$

  - Знаковый бит s определяет, является число положительным или отрицательным
  - Мантисса M – дробное число в полуинтервале [1.0,2.0).
  - Порядок E определяет степень 2 в третьем множителе

- Кодировка
  - Наибольший значащий бит s – знаковый бит s
  - Поле exp кодирует порядок E
  - Поле frac кодирует мантиссу M

© 2015 МГУ/ВМК/СП

Размеры чисел

- Одинарная точность: 32 бита. Тип – `float`.
  - Знак s 1 бит
  - Мантисса M 23 бита
  - Порядок E 8 битов
- Двойная точность: 64 бита. Тип – `double`.
  - Знак s 1 бит
  - Мантисса M 52 бита
  - Порядок E 11 битов
- Нормализация чисел
  - Нормализованное значение – порядок не принимает «крайние» значения (одни нули или одни единицы)
  - Денормализованное значение – порядок либо ноль, либо 11…11

© 2015 МГУ/ВМК/СП

Нормализованное число

- Значение: `float f = 15213.0;`

  $15213_{10} = 11101101101101_2$

  $= 1.1101101101101_2 \times 2^{13}$

- Мантисса

  M = 1.1101101101101_2

  frac = $11011011011010000000000_2$

- Порядок

  E = 13

  Смещение = 127

  Exp = E + Смещение = 140 = 10001100_2

- Итого:

0	10001100	11011011011010000000000
s	**exp**	**frac**

© 2015 МГУ/ВМК/СП

Денормализованное число

- Условие: exp = 000…0
- Значение порядка: `E = –Смещение + 1` (вместо `E = 0 – Смещение`)
- Мантисса кодируется с ведущим `0`: `M = 0.`xxx…x_2
  - xxx…x: биты поля `frac`
- Примеры
  - exp = 000…0, frac = 000…0
    - Представляет число ноль
    - Различные кодировки для +0 и –0
  - exp = 000…0, frac ≠ 000…0
    - Кодируются числа близкие к 0.0
    - Распределены по числовой прямой с равным шагом

© 2015 МГУ/ВМК/СП

Особые числа

- Условие: exp = 111…1
- Пример: exp = 111…1, frac = 000…0
  - Представляет бесконечно большое число ∞ (как положительное, так и отрицательное)
  - Требуются для операций в которых может произойти переполнение

    `1.0/0.0 = –1.0/–0.0 = +∞`

    `1.0/–0.0 = –∞`
- Пример: exp = 111…1, frac ≠ 000…0
  - Not-a-Number (NaN)
  - Используется в ситуациях, когда значение операции не определено

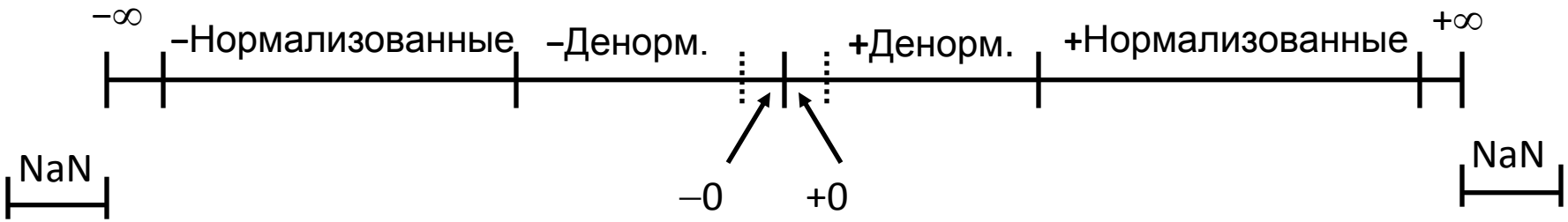
    `sqrt(–1)`

    `∞ – ∞`

    `∞ × 0`

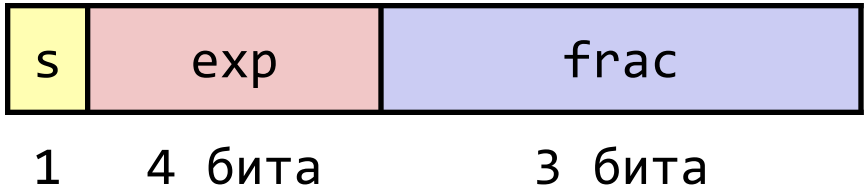
© 2015 МГУ/ВМК/СП

Диапазоны значений



© 2015 МГУ/ВМК/СП

Пример



- 8-разрядные числа с плавающей точкой
  - Знаковый бит – старший бит
  - Следующие четыре бита – порядок, смещение – 7
  - Последние три бита – дробная часть (мантисса)

- Выполнены все требования стандарта IEEE 754 к формату числа
  - Реализованы нормализованные и денормализованные числа
  - Представлены значения 0, NaN, бесконечность

© 2015 МГУ/ВМК/СП

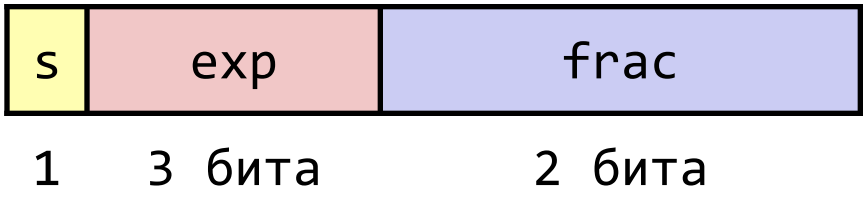
Диапазоны значений (только для положительных чисел)

	s	exp	frac	E	Значения	
	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	Ближайшее к 0
	0	0000	010	-6	2/8*1/64 = 2/512	
Денормализованные числа	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	Наибольшее денорм.
	0	0001	000	-6	8/8*1/64 = 8/512	Наименьшее норм.
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	Ближайшее к 1 «снизу»
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	Ближайшее к 1 «сверху»
	0	0111	010	0	10/8*1 = 10/8	
Нормализованные числа	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	Наибольшее норм.
	0	1111	000	n/a	inf	

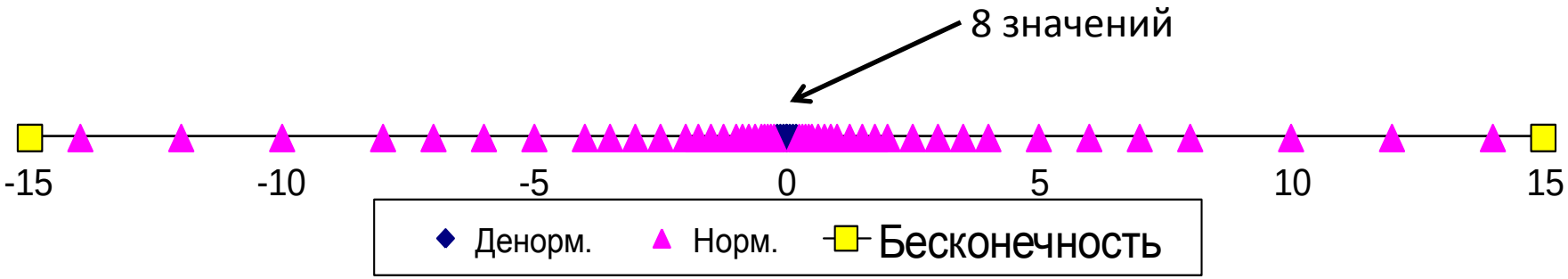
© 2015 МГУ/ВМК/СП

Распределение значений по числовой прямой

- 6-разрядный формат
  - e = 3 бита порядка
  - f = 2 бита мантиссы
  - Смещение $2^{3-1}-1 = 3$



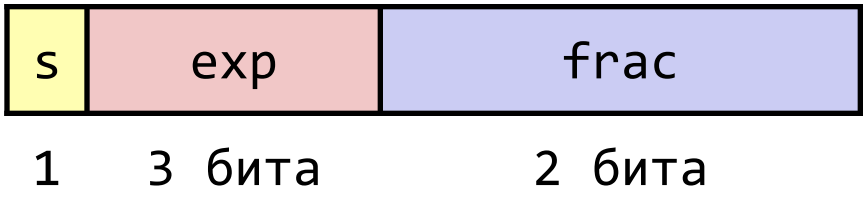
- Распределение сильно «сгущается» в окрестности 0

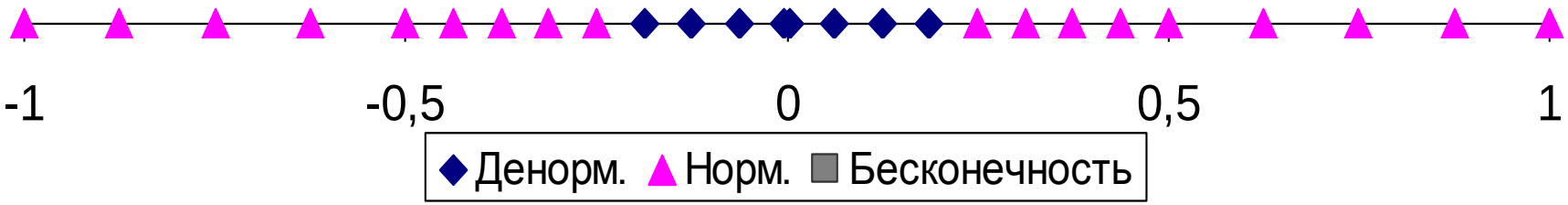


© 2015 МГУ/ВМК/СП

Распределение значений по числовой прямой (вид вблизи)

- 6-разрядный формат
  - e = 3 бита порядка
  - f = 2 бита мантиссы
  - Смещение 3





© 2015 МГУ/ВМК/СП

Некоторые числа

Описание	*exp*	*frac*	*Численное значение*
• Ноль	00…00	00…00	0.0
• Наименьшее «+» денорм.	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
– Одинарная точность ≈ 1.4×10^{-45}			
– Двойная точность ≈ 4.9×10^{-324}			
• Наибольшее денорм.	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
– Одинарная точность ≈ 1.18×10^{-38}			
– Двойная точность ≈ 2.2×10^{-308}			
• Наименьшее «+» норм.	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
– Немногим больше чем наибольшее денормализованное			
• Единица	01…11	00…00	1.0
• Наибольшее норм.	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
– Одинарная точность ≈ 3.4×10^{38}			
– Двойная точность ≈ 1.8×10^{308}			

Точность {одинарная,двойная}

© 2015 МГУ/ВМК/СП

Особенности кодировки

- FP ноль совпадает с целочисленным нулем
  - Все биты = 0
- Допустимо (в большинстве случаев) использовать беззнаковое целочисленное сравнение
  - Сперва сравниваем знаковые биты
  - Необходимо рассматривать −0 = 0
  - NaNs
    - В целочисленной интерпретации больше, чем любые другие числа
    - Что необходимо выдавать в качестве результата сравнения?
  - В противном случае …
    - Денормализованные vs. Нормализованные
    - Нормализованные vs. Бесконечность

© 2015 МГУ/ВМК/СП

Операции над числами с плавающей точкой

- $x +_f y = \text{Round}(x + y)$
- $x \times_f y = \text{Round}(x \times y)$
- Основная идея
  - Сперва вычислить точный результат
  - Поместить результат в требуемый размер точности
    - Переполнение, если порядок слишком большой
    - Возможно придется округлять поле frac

© 2015 МГУ/ВМК/СП

Округление

- Способы округления

	1.40	1.60	1.50	2.50	–1.50
– К нулю	1	1	1	2	–1
– К наименьшему (−∞)	1	1	1	2	–2
– К наибольшему (+∞)	2	2	2	3	–1
– К ближайшему (✓)	1	2	2	2	–2

© 2015 МГУ/ВМК/СП

Округление к ближайшему целому числу

- Основной способ округления
  - Все остальные способы дают статистическое смещение
    - Пример: суммирование положительных чисел будет давать устойчивую недо- или пере- оценку результата
- Применимо при округлении в произвольной позиции дроби
  - Когда число расположено точно посредине двух значений к которым можно округлить
    - Округляют к тому числу, у которого наименьшая значащая цифра четная
  - Например, округление до ближайших сотых

1.2349999	1.23	
1.2350001	1.24	
1.2350000	1.24	(середина — округляем к большему)
1.2450000	1.24	(середина — округляем к меньшему)

© 2015 МГУ/ВМК/СП

Округление двоичных чисел

- Двоичные дробные числа
  - “Четные” числа у которых младший значащий бит 0
  - “Середина” – когда биты справа от позиции к которой происходит округление = $100\ldots_2$

- Примеры
  - Округление до ближайшей 1/4 (2 бита справа от бинарной точки)

Число	Двоичное	Окр.	Действие	Окр. число
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

© 2015 МГУ/ВМК/СП

Умножение

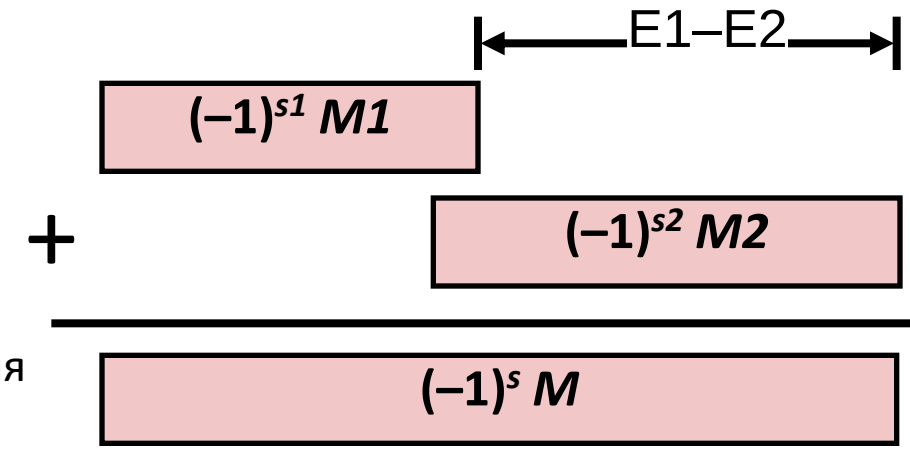
- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \ x\ \ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Точный результат: $(-1)^{s}\ M\ \ 2^{E}$
  - Знаковый бит s: s1 ^ s2
  - Мантисса M: M1 x M2
  - Порядок E: E1 + E2

- Исправление
  - Если *M* ≥ 2, сдвигаем *M* вправо (делим на 2), увеличивая *E*
  - Если *E* выходит за пределы, переполнение
  - Округляем *M* до соответствующего размера поля frac

© 2015 МГУ/ВМК/СП

Сложение

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
  - Пусть E1 > E2
- Точный результат: $(-1)^{s}\ M\ \ 2^{E}$
  - Знаковый бит s, мантисса M:
    - Результат выравнивания и сложения
  - Порядок E: E1



- Исправление
  - Если M ≥ 2, сдвигаем M вправо, увеличивая E
  - Если M < 1, сдвигаем M влево на k позиций, уменьшая E на k
  - Переполнение если E выходит за пределы
  - Округляем *M* до соответствующего размера поля `frac`

© 2015 МГУ/ВМК/СП

Математические свойства сложения

- Выполняются ли свойства Абелевых групп
  - Замкнутость? — Да
    - Результатом может быть бесконечность или NaN
  - Коммутативность? — Да
  - Ассоциативность? — Нет
    - Переполнения и изменение результата при округлении
  - `0.0` – нейтральный элемент? — Да
  - Каждый элемент имеет обратный — Почти всегда
    - За исключением бесконечности и NaN
- Монотонность
  - $a \geq b \Rightarrow a+c \geq b+c$? — Почти всегда
    - За исключением бесконечности и NaN

© 2015 МГУ/ВМК/СП

Математические свойства умножения

- Выполняются ли свойства коммутативных колец
  - Замкнуто ли относительно умножения? Да
    - Результат может быть бесконечность или NaN
  - Умножение коммутативно? Да
  - Умножение ассоциативно? Нет
    - Возможность переполнения, неточности округления
  - 1.0 – мультипликативная единица? Да
  - Умножение дистрибутивно над сложением? Нет
    - Возможность переполнения, неточности округления

- Монотонность
  - $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$? Почти всегда
    - Исключение – бесконечность и NaN

© 2015 МГУ/ВМК/СП

Числа с плавающей точкой в языке Си

- Язык Си вводит два уровня точности
  - `float` одинарная точность
  - `double` двойная точность
- Приведение типа
  - Приведение типа между `int`, `float`, и `double` включает изменение битового представления
  - `double/float` → `int`
    - Отбрасывается дробная часть (аналогично округлению к нулю)
    - Поведение не определено, когда значение вне допустимого диапазона или NaN: как правило устанавливается `TMin`
  - `int` → `double`
    - Точное приведение, поскольку `long` и `int` 32 бита ≤ 53 бита
  - `int` → `float`
    - Будет округляться согласно принятым соглашениям

© 2015 МГУ/ВМК/СП

Задачи

- Для каждого Си-выражения объяснить:
  - почему оно верно для любого значения переменных, …
  - … либо почему ложно

```
int x = …;
float f = …;
double d = …;
```

Предполагается, что
d и f не являются NaN

- `x == (int)(double) x`
- `x == (int)(float) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0` ⇒ `((d*2) < 0.0)`
- `d > f` ⇒ `-f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

© 2015 МГУ/ВМК/СП

Числа с плавающей точкой: промежуточные итоги

- IEEE 754 – четкое определение математических свойств
- Представляются числа вида $M \times 2^E$
- Семантика операций не зависит от особенностей аппаратуры
  - Сперва точное вычисление, затем округление
- Отличия от «настоящей» арифметики
  - Нарушаются свойства ассоциативности и дистрибутивности
  - Создаются сложности для компилятора и серьезных математических вычислений